```
                                                    // operation.

            // Read in the globalID of the object to be locked.
            int globalID = inputStream.readInt();

            // Synchronize on the locks table in order to ensure thread-
            // safety.
            synchronized (locks){

               // Check for an existing owner of this lock.
               LockServer lock = (LockServer) locks.get(
                  new Integer(globalID));

               if (lock == null){     // No-one presently owns this lock,
                                      // so acquire it.

                  locks.put(new Integer(globalID), this);

                  acquireLock();      // Signal to the client the
                                      // successful acquisition of this
                                      // lock.

               }else{                 // Already owned. Append ourselves
                                      // to end of queue.

                  // Search for the end of the queue. (Implemented as
                  // linked-list)
                  while (lock.next != null){
                     lock = lock.next;
                  }

                  lock.next = this;  // Append this lock request at end.
               }

            }

         }else if (command == RELEASE_LOCK){       // This is a
                                                   // RELEASE_LOCK
                                                   // operation.

            // Read in the globalID of the object to be locked.
            int globalID = inputStream.readInt();

            // Synchronize on the locks table in order to ensure thread-
            // safety.
```

```
            synchronized (locks){

               // Check to make sure we are the owner of this lock.
               LockServer lock = (LockServer) locks.get(
                  new Integer(globalID));

               if (lock == null){
                  throw new AssertionError("Unlocked. Release failed.");
               }else if (lock.address != this.address){
                  throw new AssertionError("Trying to release a lock "
                     + "which this client doesn't own. Release "
                     + "failed.");
               }

               lock = lock.next;
               lock.acquireLock();   // Signal to the client the
                                     // successful acquisition of this
                                     // lock.

               // Shift the linked list of pending acquisitions forward
               // by one.
               locks.put(new Integer(globalID), lock);

               // Clear stale reference.
               next = null;
            }

            releaseLock();     // Signal to the client the successful
                               // release of this lock.

         }else{         // Unknown command.
            throw new AssertionError(
               "Unknown command. Operation failed.");
         }

         // Read in the next command.
         command = inputStream.readInt();

      }

   }catch (Exception e){
      throw new AssertionError("Exception: " + e.toString());
   }finally{
      try{
         // Closing down. Cleanup this connection.
```

```
            outputStream.flush();
            outputStream.close();
            inputStream.close();
            socket.close();
        }catch (Throwable t){
            t.printStackTrace();
        }
        // Garbage these references.
        outputStream = null;
        inputStream = null;
        socket = null;
    }

)

/** Send a positive acknowledgement of an ACQUIRE_LOCK operation. */
public void acquireLock() throws IOException{
    outputStream.writeInt(ACK);
    outputStream.flush();
}

/** Send a positive acknowledgement of a RELEASE_LOCK operation. */
public void releaseLock() throws IOException{
    outputStream.writeInt(ACK);
    outputStream.flush();
}

}
```

ANNEXURE D6

LockLoader.java

This excerpt is the source-code of LockLoader.java, which modifies an application program code, such as the example.java application code of Annexure D3, as it is being loaded into a JAVA virtual machine in accordance with steps 161B, 162B, 163B, and 164B of Fig. 26. LockLoader.java makes use of the convenience classes of Annexures A12 through to A36 during the modification of a compiled JAVA classfile.

```
import java.lang.*;
import java.io.*;
import java.net.*;
```

```
public class LockLoader extends URLClassLoader{

   public LockLoader(URL[] urls){
      super(urls);
   }

   protected Class findClass(String name)
   throws ClassNotFoundException{

      ClassFile cf = null;

      try{
         BufferedInputStream in =
            new BufferedInputStream(findResource(name.replace('.',
            '/').concat(".class")).openStream());

         cf = new ClassFile(in);

      }catch (Exception e){throw new ClassNotFoundException(e.toString());}

      // Class-wide pointers to the enterindex and exitindex.
      int enterindex = -1;
      int exitindex = -1;

      for (int i=0; i<cf.methods_count; i++){
         for (int j=0; j<cf.methods[i].attributes_count; j++){
            if (!(cf.methods[i].attributes[j] instanceof Code_attribute))
               continue;

            Code_attribute ca = (Code_attribute)
cf.methods[i].attributes[j];

            boolean changed = false;

            for (int z=0; z<ca.code.length; z++){
               if ((ca.code[z][0] & 0xff) == 194){    // Opcode for a
                                                      // MONITORENTER
                                                      // instruction.
                  changed = true;

                  // Next, realign the code array, making room for the
                  // insertions.
                  byte[][] code2 = new byte[ca.code.length+2][];
                  System.arraycopy(ca.code, 0, code2, 0, z);
                  code2[z+1] = ca.code[z];
                  System.arraycopy(ca.code, z+1, code2, z+3,
                     ca.code.length-(z+1));
                  ca.code = code2;


                  // Next, insert the DUP instruction.
                  ca.code[z] = new byte[1];
                  ca.code[z][0] = (byte) 89;


                  // Finally, insert the INVOKESTATIC instruction.
                  if (enterindex == -1){
                     // This is the first time this class is encountering
the
                     // acquirelock instruction, so have to add it to the
                     // constant pool.
                     cp_info[] cpi = new cp_info[cf.constant_pool.length+6];
                     System.arraycopy(cf.constant_pool, 0, cpi, 0,
                        cf.constant_pool.length);
                     cf.constant_pool = cpi;
                     cf.constant_pool_count += 6;
```

```
                    CONSTANT_Utf8_info u1 =
                       new CONSTANT_Utf8_info("LockClient");
                    cf.constant_pool[cf.constant_pool.length-6] = u1;

                    CONSTANT_Class_info c1 = new CONSTANT_Class_info(
                       cf.constant_pool_count-6);
                    cf.constant_pool[cf.constant_pool.length-5] = c1;

                    u1 = new CONSTANT_Utf8_info("acquireLock");
                    cf.constant_pool[cf.constant_pool.length-4] = u1;

                    u1 = new CONSTANT_Utf8_info("(Ljava/lang/Object;)V");
                    cf.constant_pool[cf.constant_pool.length-3] = u1;

                    CONSTANT_NameAndType_info n1 =
                       new CONSTANT_NameAndType_info(
                       cf.constant_pool.length-4, cf.constant_pool.length-
3);
                    cf.constant_pool[cf.constant_pool.length-2] = n1;

                    CONSTANT_Methodref_info m1 = new
CONSTANT_Methodref_info(
                       cf.constant_pool.length-5, cf.constant_pool.length-
2);
                    cf.constant_pool[cf.constant_pool.length-1] = m1;
                    enterindex = cf.constant_pool.length-1;
                 )
                 ca.code[z+2] = new byte[3];
                 ca.code[z+2][0] = (byte) 184;
                 ca.code[z+2][1] = (byte) ((enterindex >> 8) & 0xff);
                 ca.code[z+2][2] = (byte) (enterindex & 0xff);

                 // And lastly, increase the CODE_LENGTH and
ATTRIBUTE_LENGTH
                 // values.
                 ca.code_length += 4;
                 ca.attribute_length += 4;

                 z += 1;

              }else if ((ca.code[z][0] & 0xff) == 195){    // Opcode for a
                                                           // MONITOREXIT
                                                           // instruction.
                 changed = true;

                 // Next, realign the code array, making room for the
                 // insertions.
                 byte[][] code2 = new byte[ca.code.length+2][];
                 System.arraycopy(ca.code, 0, code2, 0, z);
                 code2[z+1] = ca.code[z];
                 System.arraycopy(ca.code, z+1, code2, z+3,
                    ca.code.length-(z+1));
                 ca.code = code2;

                 // Next, insert the DUP instruction.
                 ca.code[z] = new byte[1];
                 ca.code[z][0] = (byte) 89;

                 // Finally, insert the INVOKESTATIC instruction.
                 if (exitindex == -1){
                    // This is the first time this class is encountering
the
                    // acquirelock instruction, so have to add it to the
                    // constant pool.
                    cp_info[] cpi = new cp_info[cf.constant_pool.length+6];
                    System.arraycopy(cf.constant_pool, 0, cpi, 0,
                       cf.constant_pool.length);
```

```
                    cf.constant_pool = cpi;
                    cf.constant_pool_count += 6;

                    CONSTANT_Utf8_info u1 =
                        new CONSTANT_Utf8_info("LockClient");
                    cf.constant_pool[cf.constant_pool.length-6] = u1;

                    CONSTANT_Class_info c1 = new CONSTANT_Class_info(
                        cf.constant_pool_count-6);
                    cf.constant_pool[cf.constant_pool.length-5] = c1;

                    u1 = new CONSTANT_Utf8_info("releaseLock");
                    cf.constant_pool[cf.constant_pool.length-4] = u1;

                    u1 = new CONSTANT_Utf8_info("(Ljava/lang/Object;)V");
                    cf.constant_pool[cf.constant_pool.length-3] = u1;

                    CONSTANT_NameAndType_info n1 =
                        new CONSTANT_NameAndType_info(
                        cf.constant_pool.length-4, cf.constant_pool.length-
3);
                    cf.constant_pool[cf.constant_pool.length-2] = n1;

                    CONSTANT_Methodref_info m1 = new
CONSTANT_Methodref_info(
                        cf.constant_pool.length-5, cf.constant_pool.length-
2);
                    cf.constant_pool[cf.constant_pool.length-1] = m1;
                    exitindex = cf.constant_pool.length-1;
                }
                ca.code[z+2] = new byte[3];
                ca.code[z+2][0] = (byte) 184;
                ca.code[z+2][1] = (byte) ((exitindex >> 8) & 0xff);
                ca.code[z+2][2] = (byte) (exitindex & 0xff);

                // And lastly, increase the CODE_LENGTH and
ATTRIBUTE_LENGTH
                // values.
                ca.code_length += 4;
                ca.attribute_length += 4;

                z += 1;

            }

        }

        // If we changed this method, then increase the stack size by
one.
        if (changed){
            ca.max_stack++;          // Just to make sure.
        }
    }
}


    try{

        ByteArrayOutputStream out = new ByteArrayOutputStream();
        cf.serialize(out);

        byte[] b = out.toByteArray();

        return defineClass(name, b, 0, b.length);
```

```
        }catch (Exception e){
            throw new ClassNotFoundException(name);
        }
    }
}
```

END OF ANNEXURES

CLAIMS

I/We Claim:

1. A method of running simultaneously on a plurality of computers at least one application program each written to operate on only a single computer but modified by a modification routine to run simultaneously on the plurality of computers, said computers being interconnected by means of a communications network, said method comprising the steps of:

(i) executing different portions of said application program(s) on different ones of said computers and for each said portion creating a like plurality of substantially identical objects each in the corresponding computer.

2. The method as claimed in claim 1, comprising the further step of:

(ii) naming each of said plurality of substantially identical objects with a substantially identical name.

3. The method as claimed in claim 1, wherein:

each said object having a substantially identical name.

4. The method as claimed in claim 3, comprising the further step of:

(iii) creating the initial contents of each of said identically named objects substantially the same.

5. The method as claimed in claim 3, comprising the further step of:

(iii) deleting all said identical objects collectively when all of said plurality of computers no longer need to refer to their corresponding object.

6. The method as claimed in claim 3, comprising the further step of:

(iii) requiring any of said computers wishing to utilize a named object therein to acquire an authorizing lock which permits said utilization and which prevents all the other computers from utilizing their corresponding named object until said authorizing lock is relinquished..

7. The method as claimed in claim 1, wherein (P1) each computer having an independent local memory.

8. The method as claimed in claim 7, wherein (P1) each object accessible only by the corresponding portion of said application program.

9. The method as claimed in claim 2, comprising the further step of:

(ii) naming each of said plurality of substantially identical objects with a substantially identical name.

10. The method as claimed in claim 1, wherein:

each computer having an independent local memory;

each object accessible only by the corresponding portion of said application program; and the method comprising the further steps of:

(ii) naming each of said plurality of substantially identical objects with a substantially identical name;

(iii) creating the initial contents of each of said identically named objects substantially the same;

(iv) deleting all said identical objects collectively when all of said plurality of computers no longer need to refer to their corresponding object; and

(v) requiring any of said computers wishing to utilize a named object therein to acquire an authorizing lock which permits said utilization and which prevents all the other computers from utilizing their corresponding named object until said authorizing lock is relinquished.

11. The method as claimed in claim 1, comprising the further steps of:

(ii) creating the initial contents of each of said identically named objects substantially the same.

(iii) deleting all said identical objects collectively when all of said plurality of computers no longer need to refer to their corresponding object.

(iv) requiring any of said computers wishing to utilize a named object therein to acquire an authorizing lock which permits said utilization and which

prevents all the other computers from utilizing their corresponding named object until said authorizing lock is relinquished..

12. The method as claimed in claim 11, comprising the further step of,

(v) if a portion of said application program running on one of said computers creates an object in that computer, then the created object is propagated to all of the other computers via said communications network.

13. The method as claimed in claim 12, including the further step of:

(vi) modifying said application program before, during or after loading by inserting an initialization routine to modify each instance at which said application program creates an object, said initialization routine propagating every object created by one computer to all said other computers.

14. A method as claimed in claim 11, including the further step of:

(vii) providing each said computer with a distributed run time means to communicate between said computers via said communications network.

15. A method as claimed in claim 14, including the further step of:

(viii) providing a shared table accessible by each said distributed run time means and in which is stored the identity of any computer which no longer requires to access an object, together with the identity of the object.

16. A method as claimed in claim 15, including the further step of:

(ix) associating a counter means with said shared table, said counter means storing a count of the number of said computers which no longer require to access said object.

17. A method as claimed in claim 16, including the further step of:

(x) providing an additional computer on which said shared program does not run and which hosts said shared table and counter, said additional computer being connected to said communications network.

18. A method as claimed in claim 14, including the further step of:

(xi) providing a shared table accessible by each said distributed run time means and in which is stored the identity of any computer which currently has to access an object, together with the identity of the object.

19. A method as claimed in claim 18, including the further step of:

(xii) associating a counter means with said shared table, said counter means storing a count of the number of said computers which seek access to said object.

20. A method as claimed in claim 19, including the further step of:

(xiii) providing an additional computer on which said shared program does not run and which hosts said shared table and counter, said additional computer being connected to said communications network.

21. The method as claimed in claim 8, comprising the further step of,

(ii) naming each of said plurality of substantially identical objects with a substantially identical name.

22. The method as claimed in claim 21, comprising the further step of,

(iii) if a portion of said application program running on one of said computers changes the contents of an object in that computer, then the change in content of said object is propagated to all of the other computers via said communications network to change the content of the corresponding object in each of said other computers.

23. The method as claimed in claim 22, including the further step of:

(iv) modifying said application program before, during or after loading by inserting an updating propagation routine to modify each instance at which said application program writes to memory, said updating propagation routine propagating every memory write by one computer to all said other computers.

24. The method as claimed in claim 23, including the further step of:

(v) modifying said application program utilizing a procedure selected from the group of procedures consisting of re-compilation at loading, pre-compilation prior to loading,

compilation prior to loading, just-in-time compilation, and re-compilation after loading and before execution of the relevant portion of application program.

25. The method as claimed in claim 22, including the further step of:

(vi) transferring the modified application program to all said computers utilizing a procedure selected from the group consisting of master/slave transfer, branched transfer and cascaded transfer.

26 The method as claimed in claim 4, comprising the further step of,

(iii) if a portion of said application program running on one of said computers creates an object in that computer, then the created object is propagated to all of the other computers via said communications network.

27. The method as claimed in claim 26, including the further step of:

(iv) modifying said application program before, during or after loading by inserting an initialization routine to modify each instance at which said application program creates an object, said initialization routine propagating every object created by one computer to all said other computers.

28. The method as claimed in claim 27, including the further step of:

(v) modifying said application program utilizing a procedure selected from the group of procedures consisting of re-compilation at loading, pre-compilation prior to loading, compilation prior to loading, just-in-time compilation, and re-compilation after loading and before execution of the relevant portion of application program.

29. The method as claimed in claim 27, including the further step of:

(vi) transferring the modified application program to all said computers utilizing a procedure selected from the group consisting of master/slave transfer, branched transfer and cascaded transfer.

30. A method as claimed in claim 5, including the further step of:

(iii) providing each said computer with a distributed run time means to communicate between said computers via said communications network.

31. A method as claimed in claim 30, including the further step of:

(iv) providing a shared table accessible by each said distributed run time means and in which is stored the identity of any computer which no longer requires to access an object, together with the identity of the object.

32. A method as claimed in claim 31, including the further step of:

(v) associating a counter means with said shared table, said counter means storing a count of the number of said computers which no longer require to access said object.

33. A method as claimed in claim 32, including the further step of:

(vi) providing an additional computer on which said shared program does not run and which hosts said shared table and counter, said additional computer being connected to said communications network.

34. A method as claimed in claim 6, including the further step of:

(iii) providing each said computer with a distributed run time means to communicate between said computers via said communications network.

35 A method as claimed in claim 34, including the further step of:

(iv) providing a shared table accessible by each said distributed run time means and in which is stored the identity of any computer which currently has to access an object, together with the identity of the object.

36. A method as claimed in claim 35, including the further step of:

(v) associating a counter or counter means with said shared table, said counter means storing a count of the number of said computers which seek access to said object.

37. A method as claimed in claim 36, including the further step of:

(vi) providing an additional computer on which said shared program does not run and which hosts said shared table and counter, said additional computer being connected to said communications network.

38. A computer program product comprising a set of program instructions stored in a storage medium or existing in electronic or digital form anywhere and operable to permit at least one computers to carry out the method as claimed in claims 1.

39. A multiple computer system having at least one application program each written to operate only on a single computer but modified by a modification routine and running simultaneously on a plurality of computers interconnected by a communications network, wherein portions of said application program(s) execute substantially simultaneously on different ones of said plurality of computers; and for each portion a like plurality of substantially identical objects are created, each in the corresponding one of the plurality of computers.

40. The system as claimed in claim 39, wherein each of said like plurality of substantially identical objects having a corresponding name or identifier on each of said plurality of computers.

41. The system as claimed in claim 39, wherein each of said like plurality of substantially identical objects having a substantially identical name.

42. The system as claimed in claim 39, wherein each of said plurality of substantially identical objects has a substantially identical name.

43. The system as claimed in claim 39, wherein each computer has an independent local memory accessible only by the corresponding portion of said application program(s).

44. The system as claimed in claim 39, wherein the initial contents of each of said identically named objects is substantially the same.

45. The system as claimed in claim 39, wherein all said identical objects are collectively deleted when each one of said plurality of computers no longer needs to refer to their corresponding object.

46. The system as claimed in claim 39, wherein said system including a lock means applicable to all said computers wherein any computer wishing to utilize a named object therein acquires an authorizing lock from said lock means which permits said utilization and which prevents all the other computers from utilizing their corresponding named object until said authorizing lock is relinquished.

47. The system as claimed in claim 39, wherein said portions are different portions

48. The system as claimed in claim 44, wherein each of said plurality of substantially identical objects has a substantially identical name.

49. The system as claimed in claim 39, wherein:

each of said like plurality of substantially identical objects having a substantially identical name;

the initial contents of each of said identically named objects is substantially the same;

all said identical objects are collectively deleted when each one of said plurality of computers no longer needs to refer to their corresponding object; and

said system including a lock means applicable to all said computers wherein any computer wishing to utilize a named object therein acquires an authorizing lock from said lock means which permits said utilization and which prevents all the other computers from utilizing their corresponding named object until said authorizing lock is relinquished.

50. The system as claimed in claim 49, wherein said lock means includes an acquire lock routine and a release lock routine, and both said routines are included in modifications made to said application program running on all said computers.

51. The system as claimed in claim 50, wherein said lock means further includes a shared table listing said named objects in use by any said computer, a lock status for each said object, and a queue of any pending lock acquisitions.

52. The system as claimed in claim 51, wherein said lock means is located within an additional computer not running said application program and connected to said communications network.

53. The system as claimed in claim 49, wherein each said computer includes a distributed run time means with the distributed run time means of each said computer able to communicate with all other computers whereby if a portion of said application program(s) running on one of said computers creates an object in that computer then the created object is propagated by the distributed run time means of said one computer to all the other computers.

54. The system as claimed in claim 49, wherein each said computer includes a distributed run time means with the distributed run time means of each said computer able to communicate with all other computers whereby if a portion of said application program(s) running on one of said computers no longer needs to refer to an object in that computer then the identity of the unreferenced object is transmitted by the distributed run time means of said one computer to a shared table accessible by all the other computers.

55. The system as claimed in claim 49, wherein each said application program is modified before, during, or after loading by inserting an initialization routine to modify each instance at which said application program creates an object, said initialization routine propagating every object newly created by one computer to all said other computers.

56. The system as claimed in claim 55, wherein the application program is modified in accordance with a procedure selected from the group of procedures consisting of re-compilation at loading, pre-compilation prior to loading, compilation prior to loading, just-in-time compilation, and re-compilation after loading and before execution of the relevant portion of application program.

57. The system as claimed in claim 55, wherein said modified application program is transferred to all said computers in accordance with a procedure selected from the group consisting of master/slave transfer, branched transfer and cascaded transfer.

58. The system as claimed in claim 49, wherein the local memory capacity allocated to the, or each, said application program is substantially identical and the total memory capacity available to the, or each, said application program is said allocated memory capacity.

59. The system as claimed in claim 58, wherein all said computers include a distribution update means each of which communicates via said communications link at a data transfer rate which is substantially less than the local memory read rate.

60. The system as claimed in claim 59, wherein at least some of said computers are manufactured by different manufacturers and/or have different operating systems.

61. The system as claimed in claim 43, wherein each of said plurality of substantially identical objects has a substantially identical name.

62. The system as claimed in claim 61, wherein each said computer includes a distributed run time means with the distributed run time means of each said computer able to communicate with all other computers whereby if a portion of said application program(s) running on one of said computers changes the contents of an object in that computer then the change in content for said object is propagated by the distributed run time means of said one computer to all other computers to change the content of the corresponding object in each of said other computers.

63. The system as claimed in claim 62, wherein each said application program is modified before, during, or after loading by inserting an updating propagation routine to modify each instance at which said application program writes to memory, said updating propagation routine propagating every memory write by one computer to all said other computers.

64. The system as claimed in claim 63, wherein the application program is modified in accordance with a procedure selected from the group of procedures consisting of re-compilation at loading, pre-compilation prior to loading, compilation prior to loading, just-in-time compilation, and re-compilation after loading and before execution of the relevant portion of application program.

65. The system as claimed in claim 64, wherein said modified application program is transferred to all said computers in accordance with a procedure selected from the group consisting of master/slave transfer, branched transfer and cascaded transfer.

66. The system as claimed in claim 44, wherein each said computer includes a distributed run time means with the distributed run time means of each said computer able to communicate with all other computers whereby if a portion of said application program(s) running on one of said computers creates an object in that computer then the created object is propagated by the distributed run time means of said one computer to all the other computers.

67. The system as claimed in claim 66, wherein each said application program is modified before, during, or after loading by inserting an initialization routine to modify each instance at which said application program creates an object, said initialization routine propagating every object newly created by one computer to all said other computers.

68. The system as claimed in claim 67, wherein said inserted initialization routine modifies a pre-existing initialization routine to enable the pre-existing initialization routine to execute on creation of the first of said like plurality of objects, and to disable the pre-existing initialization routine on creation of all subsequent ones of said like plurality of objects.

69. The system as claimed in claim 68, wherein the application program is modified in accordance with a procedure selected from the group of procedures consisting of re-compilation at loading, pre-compilation prior to loading, compilation prior to loading, just-in-time compilation, and re-compilation after loading and before execution of the relevant portion of application program.

70. The system as claimed in claim 66, wherein said modified application program is transferred to all said computers in accordance with a procedure selected from the group consisting of master/slave transfer, branched transfer and cascaded transfer.

71. The system as claimed in claim 45, wherein each said computer includes a distributed run time means with the distributed run time means of each said computer able to communicate with all other computers whereby if a portion of said application program(s) running on one of said computers no longer needs to refer to an object in that computer then the identity of the unreferenced object is transmitted by the distributed run time means of said one computer to a shared table accessible by all the other computers.

72. The system as claimed in claim 71, wherein each said application program is modified before, during, or after loading by inserting a finalization routine to modify each instance at which said application program no longer needs to refer to an object.

73. The system as claimed in claim 72, wherein said inserted finalization routine modified a pre-existing finalization routine to enable the pre-existing finalization routine to execute if all computers no longer need to refer to their corresponding object, and to disable the pre-existing finalization routine if at least one computer does not to refer to a corresponding object.

74. The system as claimed in claim 73, wherein the application program is modified in accordance with a procedure selected from the group of procedures consisting of re-compilation at loading, pre-compilation prior to loading, compilation prior to loading, just-in-time compilation, and re-compilation after loading and before execution of the relevant portion of application program.

75. The system as claimed in claim 71, wherein said modified application program is transferred to all said computers in accordance with a procedure selected from the group consisting of master/slave transfer, branched transfer and cascaded transfer.

76. The system as claimed in claim 46, wherein said lock means includes an acquire lock routine and a release lock routine, and both said routines are included in modifications made to said application program running on all said computers.

77. The system as claimed in claim 76, wherein said lock means further includes a shared table listing said named objects in use by any said computer, a lock status for each said object, and a queue of any pending lock acquisitions.

78. The system as claimed in claim 77, wherein said lock means is located within an additional computer not running said application program and connected to said communications network.

79. The system as claimed in claim 76, wherein each said application program is modified before, during, or after loading by inserting said acquire lock routine and said release lock routine to modify each instance at which said application program acquires and releases respectively a lock on an object.

80. The system as claimed in claim 77, wherein the application program is modified in accordance with a procedure selected from the group of procedures consisting of re-compilation at loading, pre-compilation prior to loading, compilation prior to loading, just-in-time compilation, and re-compilation after loading and before execution of the relevant portion of application program.

81. The system as claimed in claim 76, wherein said modified application program is transferred to all said computers in accordance with a procedure selected from the group consisting of master/slave transfer, branched transfer and cascaded transfer.

82. The system as claimed in claim 39, further comprising a distributed runtime component adapted to provide an exchange of program and/or data over a communications link between the plurality of computers.

83. A plurality of computers interconnected via a communications link and substantially simultaneously operating a different portion at least one application program each written to operate on only a single computer but modified by a modifier to operate substantially simultaneously coherently and consistently on said plurality of computers.

84. The plurality of computers as claimed in claim 83, wherein each said computer having an independent local memory; and each local memory being accessible only by the corresponding portion of said application program.

85. The plurality of computers as claimed in claim 84, wherein each said computer in operating said at least one application program reads and writes only to local memory physically located in each said computer, the contents of the local memory utilized by each said computer is fundamentally similar but not, at each instant, identical, and every one of said computers has distribution update means to distribute to all other said computers the content of any memory location updated by said one computer.

86. The plurality of computers as claimed in claim 85, wherein the local memory capacity allocated to the or each said application program is substantially identical and the total memory capacity available to the or each said application program is said allocated memory capacity.

87. The plurality of computers as claimed in claim 85, wherein all said distribution update means communicate via said communications link at a data transfer rate which is substantially less than the local memory read rate.

88. The plurality of computers as claimed in claim 83, wherein at least some of said computers are manufactured by different manufacturers and/or have different operating systems.

89. The plurality of computers as claimed in claim 83, wherein:

each said computer substantially simultaneously executes a different portion of said application program(s);

each said computer in operating its application program portion creates objects only in local memory physically located in each said computer; and

the contents of the local memory utilized by each said computer are fundamentally similar but not, at each instant, identical, and every one of said computers includes distribution update means or mechanism to distribute to all other said computers objects created by said one computer.

90. The plurality of computers as claimed in claim 89, wherein the local memory capacity allocated to the or each said application program is substantially identical and the total memory capacity available to the or each said application program is said allocated memory capacity.

91. The plurality of computers as claimed in claim 89, wherein all said distribution update means communicate via said communications link at a data transfer rate which is substantially less than the local memory read rate.

92. The plurality of computers as claimed in claim 89, wherein at least some of said computers are manufactured by different manufacturers and/or have different operating systems.

93. The plurality of computers as claimed in claim 83, wherein:

each said computer substantially simultaneously executes a different portion of said application program(s);

each said computer in operating its application program portion needs, or no longer needs to refer to an object only in local memory physically located in each said computer; and

the contents of the local memory utilized by each said computer is fundamentally similar but not, at each instant, identical, and every one of said computers has a finalization routine which deletes a non-referenced object only if each one of said plurality of computers no longer needs to refer to their corresponding object.

94. The plurality of computers as claimed in claim 93, wherein the local memory capacity allocated to the or each said application program is substantially identical and the total memory capacity available to the or each said application program is said allocated memory capacity.

95. The plurality of computers as claimed in claim 93, wherein all said distribution update means communicate via said communications link at a data transfer rate which is substantially less than the local memory read rate.

96. The plurality of computers as claimed in claim 93, wherein at least some of said computers are manufactured by different manufacturers and/or have different operating systems.

97. The plurality of computers as claimed in claim 93, wherein:

each said computer substantially simultaneously executes a different portion of said application program(s);

each said computer in operating its application program portion utilizes an object only in local memory physically located in each said computer; and

the contents of the local memory utilized by each said computer is fundamentally similar but not, at each instant, identical, and every one of said computers has an acquire lock routine and a release lock routine which permit utilization of the local object only by one computer and each of the remainder of said plurality of computers is locked out of utilization of their corresponding object.

98. The plurality of computers as claimed in claim 97, wherein the local memory capacity allocated to the or each said application program is substantially identical and the total memory capacity available to the or each said application program is said allocated memory capacity.

99. The plurality of computers as claimed in claim 97, wherein all said distribution update means communicate via said communications link at a data transfer rate which is substantially less than the local memory read rate.

100. The plurality of computers as claimed in claim 97, wherein at least some of said computers are manufactured by different manufacturers and/or have different operating systems.

101. A method of loading an application program written to operate only on a single computer onto each of a plurality of computers, the computers being interconnected via a communications link, and different portions of said application program(s) being substantially simultaneously executable on different computers with each computer having an independent local memory accessible only by the corresponding portion of said application program(s), the

method comprising the step of: modifying the application before, during, or after loading and before execution of the relevant portion of the application program.

102. The method as claimed in claim 101, wherein the modification of the application is different for different computers.

103. The method as claimed in claim 101, wherein said modifying step comprises:-

(i) detecting instructions which share memory records utilizing one of said computers,

(ii) listing all such shared memory records and providing a naming tag for each listed memory record,

(iii) detecting those instructions which write to, or manipulate the contents of, any of said listed memory records, and

(iv) generating an updating propagation routine corresponding to each said detected write or manipulate instruction, said updating propagation routine forwarding the re-written or manipulated contents and name tag of each said re-written or manipulated listed memory record to all of the others of said computers.

104. A method of operating simultaneously on a plurality of computers all interconnected via a communications link at least one application program each written to operate on only a single computer, each of said computers having at least a minimum predetermined local memory capacity, different portions of said application program(s) being substantially simultaneously executed on different ones of said computers with the local memory of each computer being only accessible by the corresponding portion of said application program(s), said method comprising the steps of:

(i) initially providing each local memory in substantially identical condition,

(ii) satisfying all memory reads and writes generated by each said application program portion from said corresponding local memory, and

(iii) communicating via said communications link all said memory writes at each said computer which take place locally to all the remainder of said plurality of computers whereby the contents of the local memory utilised by each said computer, subject to an updating data transmission delay, remains substantially identical.

105. The method as claimed in claim 104, including the further step of:

(iv) communicating said local memory writes constituting an updating data transmission at a data transfer rate which is substantially less than the local memory read rate.

106. A method of compiling or modifying an application program written to operate on only a single computer but to run simultaneously on a plurality of computers interconnected via a communications link, with different portions of said application program(s) executing substantially simultaneously on different ones of said computers each of which has an independent local memory accessible only by the corresponding portion of said application program, said method comprising the steps of:

(i) detecting instructions which share memory records utilizing one of said computers,

(ii) listing all such shared memory records and providing a naming tag for each listed memory record,

(iii) detecting those instructions which write to, or manipulate the contents of, any of said listed memory records, and

(iv) activating an updating propagation routine following each said detected write or manipulate instruction, said updating propagation routine forwarding the re-written or manipulated contents and name tag of each said re-written or manipulated listed memory record to the remainder of said computers.

107. The method as claimed in claim 106, and carried out prior to loading the application program onto each said computer, or during loading of the application program onto each said computer, or after loading of the application program onto each said computer and before execution of the relevant portion of the application program.

108. A method of compiling or modifying an application program written to operate on only a single computer to have different portions thereof to execute substantially simultaneously on different ones of a plurality of computers interconnected via a communications link, said method comprising the steps of:

(i) detecting instructions which create objects utilizing one of said computers,

(ii) activating an initialization routine following each said detected object creation instruction, said initialization routine forwarding each created object to the remainder of said computers.

109. The method as claimed in claim 108, and carried out prior to loading the application program onto each said computer, or during loading of the application program onto each said computer, or after loading of the application program onto each said computer and before execution of the relevant portion of the application program.

110. In a multiple thread processing computer operation in which individual threads of a single application program written to operate on only a single computer are modified and simultaneously being processed each on a different corresponding one of a plurality of computers, each being interconnected via a communications link, the improvement comprising at least one of:

communicating information in local memory physically associated with the computer processing each thread to the local memory of each other said computer via said communications link;

communicating changes in the contents of local memory physically associated with the computer processing each thread to the local memory of each other said computer via said communications link;

communicating objects created in local memory physically associated with the computer processing each thread to the local memory of each other said computer via said communications link;

collectively deleting all said corresponding objects when each one of said plurality of computers no longer needs to refer to their corresponding object; and

permitting only one of said computers to utilize an object and preventing all the remaining computers from simultaneously utilizing their corresponding object.

111. The improvement as claimed in claim 110, wherein the improvement comprises all of:

communicating changes in the contents of local memory physically associated with the computer processing each thread to the local memory of each other said computer via said communications link;

communicating objects created in local memory physically associated with the computer processing each thread to the local memory of each other said computer via said communications link;

collectively deleting all said corresponding objects when each one of said plurality of computers no longer needs to refer to their corresponding object; and

permitting only one of said computers to utilize an object and preventing all the remaining computers from simultaneously utilizing their corresponding object.

112. The improvement as claimed in claim 110, wherein the improvement comprising: communicating information in local memory physically associated with the computer processing each thread to the local memory of each other said computer via said communications link.

113. The improvement as claimed in claim 112, wherein each of the plurality of computers having an independent local memory accessible only by the corresponding thread; and the communicating information comprises communicating changes in the contents of local memory physically associated with the computer processing each thread to the local memory of each other said computer via said communications link.

114. The improvement as claimed in claim 113, wherein changes to the memory associated with one said thread are communicated by the computer of said one thread to all other said computers.

115. The improvement as claimed in claim 113, wherein changes to the memory associated with one said thread are transmitted to the computer associated with another said thread and are transmitted thereby to all said other computers.

116. The improvement as claimed in claim 112, wherein the communicating information comprises communicating objects created in local memory physically associated with the computer processing each thread to the local memory of each other said computer via said communications link.

117. The improvement as claimed in claim 116, wherein objects created in the memory associated with one said thread are communicated by the computer of said one thread to all other said computers.

118. The improvement as claimed in claim 116, wherein objects created the memory associated with one said thread are transmitted to the computer associated with another said thread and are transmitted thereby to all said other computers.

119. The improvement as claimed in claim 110, wherein the improvement comprising:

collectively deleting all said corresponding objects when each one of said plurality of computers no longer needs to refer to their corresponding object.

120. The improvement as claimed in claim 119, wherein an object residing in the memory associated with one said thread and to be deleted has its identity communicated by the computer of said one thread to a shared table accessible by all other said computers.

121. The improvement as claimed in claim 119, wherein an object residing in the memory associated with one said thread and to be deleted has its identity transmitted to the computer associated with another said thread and is transmitted thereby to a shared table accessible by all said other computers.

122. The improvement as claimed in claim 110, wherein the improvement comprising:

permitting only one of said computers to utilize an object and preventing all the remaining computers from simultaneously utilizing their corresponding object.

123. The improvement as claimed in claim 122, wherein an object residing in the memory associated with one said thread and to be utilized has its identity communicated by the computer of said one thread to a shared table accessible by all other said computers.

124. The improvement as claimed in claim 122, wherein an object residing in the memory associated with one said thread and to be utilized has its identity transmitted to the computer associated with another said thread and is transmitted thereby to a shared table accessible by all said other computers.

125. A method of ensuring consistent initialization of an application program written to operate on only a single computer but different portions of which are to be executed simultaneously each on a different one of a plurality of computers interconnected via a communications network, said method comprising the steps of:

(i) scrutinizing said application program at, or prior to, or after loading to detect each program step defining an initialization routine, and

(ii) modifying said initialization routine to ensure consistent operation of all said computers.

126. The method as claimed in claim 125, wherein said initialization routine is modified to execute once only on the creation of a first object by any one of said computers and is modified to be disabled on the creation of each subsequent peer copy of said object by the remainder of said computers.

127. The method claimed in claim 125, wherein step (ii) comprises the steps of:

(iii) loading and executing said initialization routine on one of said computers,

(iv) modifying said initialization routine by said one computer, and

(v) transferring said modified initialization routine to each of the remaining computers.

128. The method claimed in claim 126, wherein step (ii) comprises the steps of:

(iii) loading and executing said initialization routine on one of said computers,

(iv) modifying said initialization routine by said one computer, and

(v) transferring said modified initialization routine to each of the remaining computers.

129. The method as claimed in claim 127, wherein said modified initialization routine is supplied by said one computer direct to each of said remaining computers.

130. The method as claimed in claim 128, wherein said modified initialization routine is supplied in cascade fashion from said one computer sequentially to each of said remaining computers.

131. The method as claimed in claim 128, wherein said modified initialization routine is supplied by said one computer direct to each of said remaining computers.

132. The method as claimed in claim 128, wherein said modified initialization routine is supplied in cascade fashion from said one computer sequentially to each of said remaining computers.

133. The method claimed in claim 125, wherein step (ii) comprises the steps of:

(vi) loading and modifying said initialization routine on one of said computers,

(vii) said one computer sending said unmodified initialization routine to each of the remaining computers, and

(viii) each of said remaining computers modifying said initialization routine after receipt of same.

134. The method claimed in claim 126, wherein step (ii) comprises the steps of:

(vi) loading and modifying said initialization routine on one of said computers,

(vii) said one computer sending said unmodified initialization routine to each of the remaining computers, and

(viii) each of said remaining computers modifying said initialization routine after receipt of same.

135. The method claimed in claim 134, wherein said unmodified initialization routine is supplied by said one computer directly to each of said remaining computers.

136. The method claimed in claim 134, wherein said unmodified initialization routine is supplied in cascade fashion from said one computer sequentially to each of said remaining computers.

137. The method claimed in claim 135, wherein said unmodified initialization routine is supplied by said one computer directly to each of said remaining computers.

138 The method claimed in claim 135, wherein said unmodified initialization routine is supplied in cascade fashion from said one computer sequentially to each of said remaining computers.

139. A method of ensuring consistent finalization of an application program written to operate only on a single computer but different portions of which are to be executed substantially simultaneously each on a different one of a plurality of computers interconnected via a communications network, said method comprising the steps of:

(i) scrutinizing said application program at, or prior to, or after loading to detect each program step defining an finalization routine, and

(ii) modifying said finalization routine to ensure collective deletion of corresponding objects in all said computers only when each one of said computers no longer needs to refer to their corresponding object.

140. The method as claimed in claim 139, wherein said finalization routine is modified to execute to clean-up an object once only and on only one of said computers and when all of said computers no longer need to refer to said object.

141. The method claimed in claim 139, wherein step (ii) comprises the steps of:

(iii) loading and executing said finalization routine on one of said computers,

(iv) modifying said finalization routine by said one computer, and

(v) transferring said modified finalization routine to each of the remaining computers.

142. The method claimed in claim 140, wherein step (ii) comprises the steps of:

(iii) loading and executing said finalization routine on one of said computers,

(iv) modifying said finalization routine by said one computer, and

(v) transferring said modified finalization routine to each of the remaining computers.

143. The method as claimed in claim 141, wherein said modified finalization routine is supplied by said one computer direct to each of said remaining computers.

144. The method as claimed in claim 141, wherein said modified finalization routine is supplied in cascade fashion from said one computer sequentially to each of said remaining computers.

145. The method claimed in claim 139, wherein step (ii) comprises the steps of:

(vi) loading and modifying said finalization routine on one of said computers,

(vii) said one computer sending said unmodified finalization routine to each of the remaining computers, and

(viii) each of said remaining computers modifying said finalization routine after receipt of same.

146. The method claimed in claim 140, wherein step (ii) comprises the steps of:

(vi) loading and modifying said finalization routine on one of said computers,

(vii) said one computer sending said unmodified finalization routine to each of the remaining computers, and

(viii) each of said remaining computers modifying said finalization routine after receipt of same.

147. The method claimed in claim 145, wherein said unmodified finalization routine is supplied by said one computer directly to each of said remaining computers.

148. The method claimed in claim 145, wherein said unmodified finalization routine is supplied in cascade fashion from said one computer sequentially to each of said remaining computers.

149. The method as claimed in claim 139, including the further step of:

(ix) modifying said application program utilizing a procedure selected from the group of procedures consisting of re-compilation at loading, pre-compilation prior to loading, compilation prior to loading, just-in-time compilation, and re-compilation after loading and before execution of the relevant portion of application program.

150. The method as claimed in claim 139, including the further step of:

(x) transferring the modified application program to all said computers utilizing a procedure selected from the group consisting of master/slave transfer, branched transfer and cascaded transfer.

151. The method as claimed in claim 140, including the further step of:

(ix) modifying said application program utilizing a procedure selected from the group of procedures consisting of re-compilation at loading, pre-compilation prior to loading, compilation prior to loading, just-in-time compilation, and re-compilation after loading and before execution of the relevant portion of application program.

152. The method as claimed in claim 140, including the further step of:

(x) transferring the modified application program to all said computers utilizing a procedure selected from the group consisting of master/slave transfer, branched transfer and cascaded transfer.

153. A method of ensuring consistent synchronization of an application program written to operate only on a single computer but different portions of which are to be executed substantially simultaneously each on a different one of a plurality of computers interconnected via a communications network, said method comprising the steps of:

(i) scrutinizing said application program at, or prior to, or after loading to detect each program step defining an synchronization routine, and

(ii) modifying said synchronization routine to ensure utilization of an object by only one computer and preventing all the remaining computers from simultaneously utilizing their corresponding objects.

154. The method claimed in claim 153, wherein step (ii) comprises the steps of:

(iii) loading and executing said synchronization routine on one of said computers,

(iv) modifying said synchronization routine by said one computer, and

(v) transferring said modified synchronization routine to each of the remaining computers.

155. The method as claimed in claim 154, wherein said modified synchronization routine is supplied by said one computer direct to each of said remaining computers.

156. The method as claimed in claim 154, wherein said modified synchronization routine is supplied in cascade fashion from said one computer sequentially to each of said remaining computers.

157. The method claimed in claim 154, wherein step (ii) comprises the steps of:

(vi) loading and modifying said synchronization routine on one of said computers,

(vii) said one computer sending said unmodified synchronization routine to each of the remaining computers, and

(viii) each of said remaining computers modifying said synchronization routine after receipt of same.

158. The method claimed in claim 157, wherein said unmodified synchronization routine is supplied by said one computer directly to each of said remaining computers.

159. The method claimed in claim 157, wherein said unmodified synchronization routine is supplied in cascade fashion from said one computer sequentially to each of said remaining computers.

160. The method as claimed in claim 153, including the further step of:

(ix) modifying said application program utilizing a procedure selected from the group of procedures consisting of re-compilation at loading, pre-compilation prior to loading, compilation prior to loading, just-in-time compilation, and re-compilation after loading and before execution of the relevant portion of application program.

161. The method as claimed in claim 153, including the further step of:

(x) transferring the modified application program to all said computers utilizing a procedure selected from the group consisting of master/slave transfer, branched transfer and cascaded transfer.

162. A computer program product comprising a set of program instructions stored in a storage medium and operable to permit a plurality of computers to carry out the method as claimed in claim 1.

163. A plurality of computers interconnected via a communication network and operable to ensure consistent initialization of an application program written to operate only on a single computer but running simultaneously on said computers, said computers being programmed to carry out the method as claimed in claim 1 or being loaded with the computer program product as claimed in claim 162.

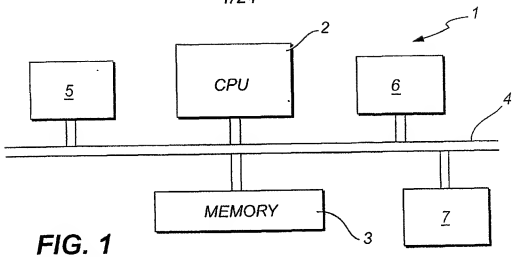

FIG. 1
PRIOR ART

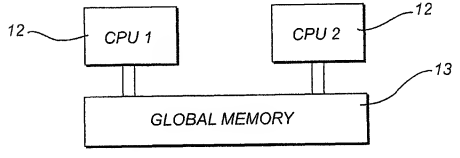

FIG. 2
PRIOR ART

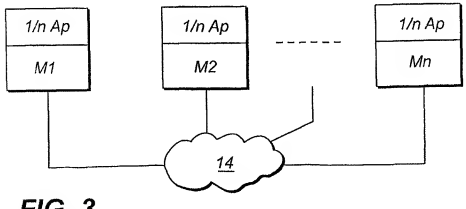

FIG. 3
PRIOR ART

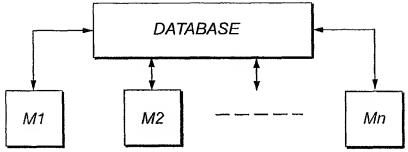

FIG. 4
PRIOR ART

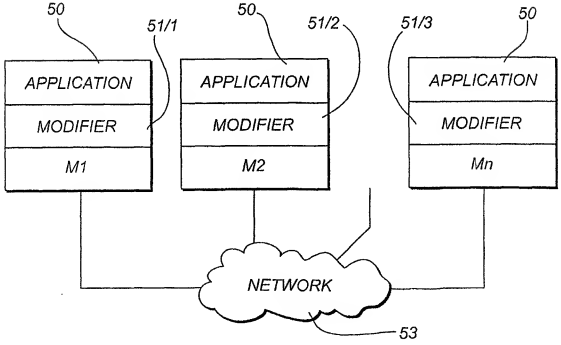

FIG. 5

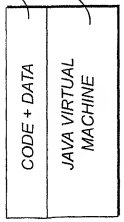

FIG. 6
PRIOR ART

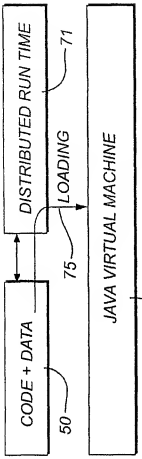

FIG. 7

FIG. 8

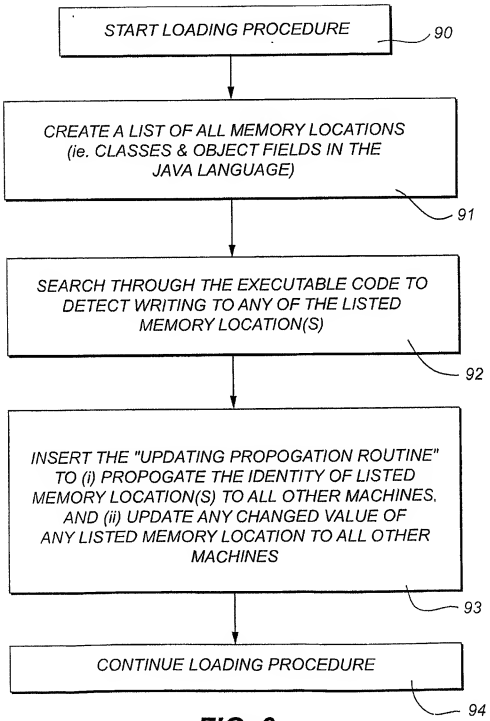
START LOADING PROCEDURE
90
CREATE A LIST OF ALL MEMORY LOCATIONS (ie. CLASSES & OBJECT FIELDS IN THE JAVA LANGUAGE)
91
SEARCH THROUGH THE EXECUTABLE CODE TO DETECT WRITING TO ANY OF THE LISTED MEMORY LOCATION(S)
92
INSERT THE "UPDATING PROPOGATION ROUTINE" TO (i) PROPOGATE THE IDENTITY OF LISTED MEMORY LOCATION(S) TO ALL OTHER MACHINES, AND (ii) UPDATE ANY CHANGED VALUE OF ANY LISTED MEMORY LOCATION TO ALL OTHER MACHINES
93
CONTINUE LOADING PROCEDURE
94

FIG. 9

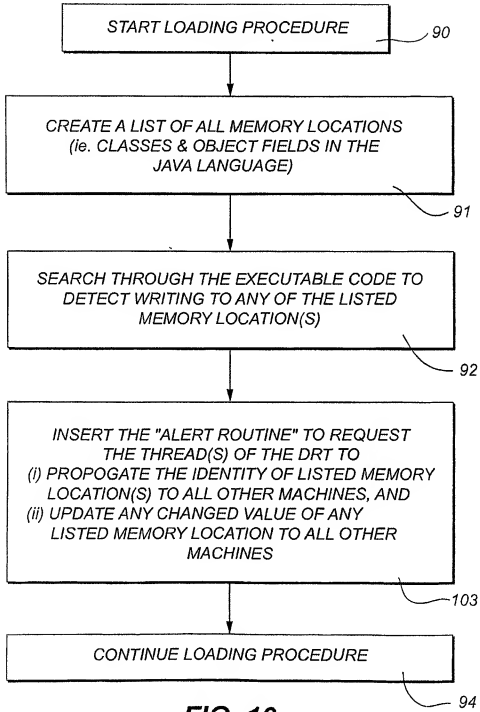
START LOADING PROCEDURE
90
CREATE A LIST OF ALL MEMORY LOCATIONS (ie. CLASSES & OBJECT FIELDS IN THE JAVA LANGUAGE)
91
SEARCH THROUGH THE EXECUTABLE CODE TO DETECT WRITING TO ANY OF THE LISTED MEMORY LOCATION(S)
92
INSERT THE "ALERT ROUTINE" TO REQUEST THE THREAD(S) OF THE DRT TO (i) PROPOGATE THE IDENTITY OF LISTED MEMORY LOCATION(S) TO ALL OTHER MACHINES, AND (ii) UPDATE ANY CHANGED VALUE OF ANY LISTED MEMORY LOCATION TO ALL OTHER MACHINES
103
CONTINUE LOADING PROCEDURE
94

FIG. 10

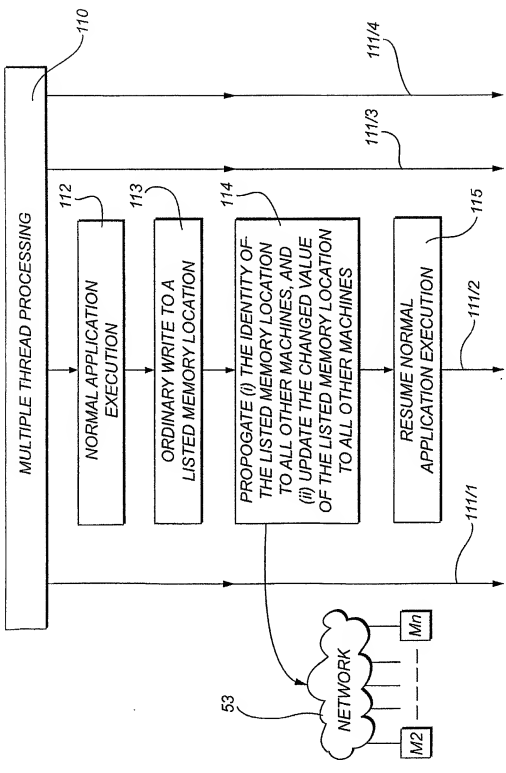
MULTIPLE THREAD PROCESSING
110
NORMAL APPLICATION EXECUTION
112
ORDINARY WRITE TO A LISTED MEMORY LOCATION
113
PROPOGATE (i) THE IDENTITY OF THE LISTED MEMORY LOCATION TO ALL OTHER MACHINES, AND (ii) UPDATE THE CHANGED VALUE OF THE LISTED MEMORY LOCATION TO ALL OTHER MACHINES
114
RESUME NORMAL APPLICATION EXECUTION
115
111/1
111/2
111/3
111/4
NETWORK
53
M2
Mn

FIG.11

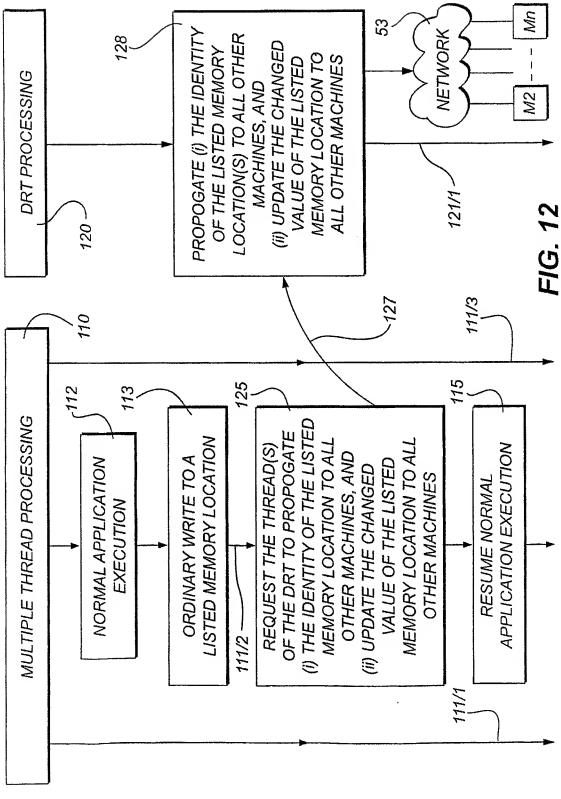
MULTIPLE THREAD PROCESSING
110
NORMAL APPLICATION EXECUTION
112
ORDINARY WRITE TO A LISTED MEMORY LOCATION
113
111/2
REQUEST THE THREAD(S) OF THE DRT TO PROPOGATE (i) THE IDENTITY OF THE LISTED MEMORY LOCATION TO ALL OTHER MACHINES, AND (ii) UPDATE THE CHANGED VALUE OF THE LISTED MEMORY LOCATION TO ALL OTHER MACHINES
125
RESUME NORMAL APPLICATION EXECUTION
115
111/1
111/3
127
DRT PROCESSING
120
PROPOGATE (i) THE IDENTITY OF THE LISTED MEMORY LOCATION(S) TO ALL OTHER MACHINES, AND (ii) UPDATE THE CHANGED VALUE OF THE LISTED MEMORY LOCATION TO ALL OTHER MACHINES
128
121/1
53
NETWORK
M2
Mn

FIG. 12

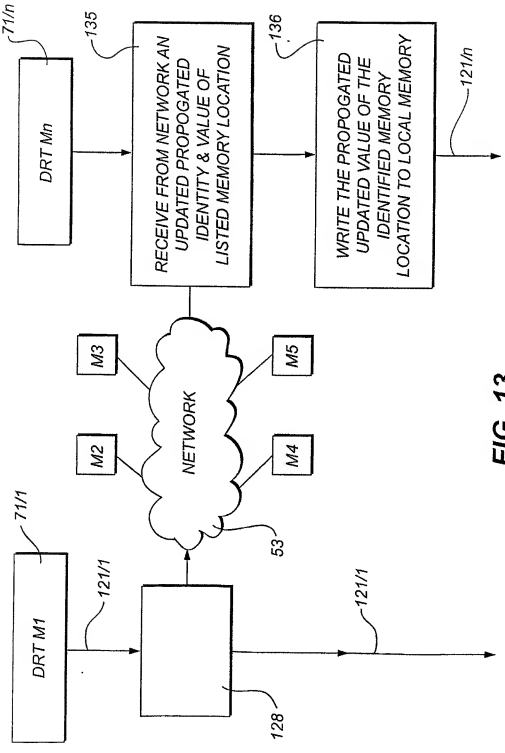
71/n
DRT Mn
135
RECEIVE FROM NETWORK AN UPDATED PROPOGATED IDENTITY & VALUE OF LISTED MEMORY LOCATION
136
WRITE THE PROPOGATED UPDATED VALUE OF THE IDENTIFIED MEMORY LOCATION TO LOCAL MEMORY
121/n
M3
M5
NETWORK
M2
M4
53
71/1
DRT M1
121/1
121/1
128

FIG. 13

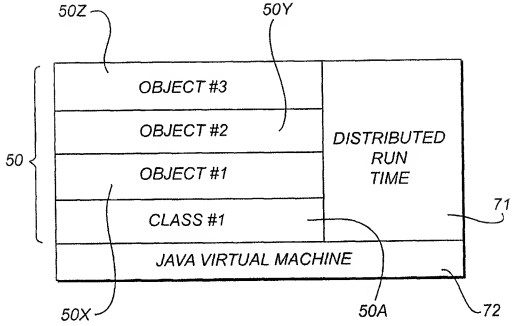

FIG. 14
PRIOR ART

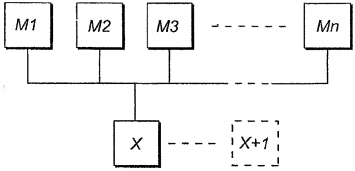
M1
M2
M3
Mn
X
X+1

FIG. 15

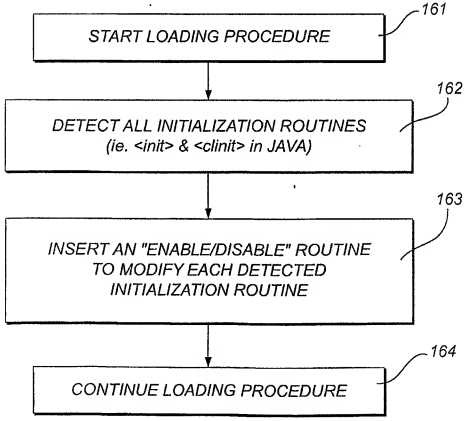
START LOADING PROCEDURE
161
DETECT ALL INITIALIZATION ROUTINES
(ie. <init> & <clinit> in JAVA)
162
INSERT AN "ENABLE/DISABLE" ROUTINE
TO MODIFY EACH DETECTED
INITIALIZATION ROUTINE
163
CONTINUE LOADING PROCEDURE
164

FIG. 16

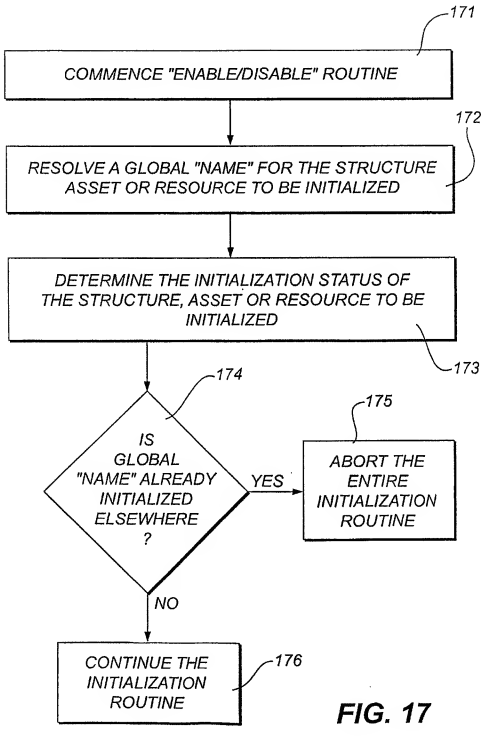
171
COMMENCE "ENABLE/DISABLE" ROUTINE
172
RESOLVE A GLOBAL "NAME" FOR THE STRUCTURE ASSET OR RESOURCE TO BE INITIALIZED
DETERMINE THE INITIALIZATION STATUS OF THE STRUCTURE, ASSET OR RESOURCE TO BE INITIALIZED
173
174
IS GLOBAL "NAME" ALREADY INITIALIZED ELSEWHERE ?
YES
175
ABORT THE ENTIRE INITIALIZATION ROUTINE
NO
CONTINUE THE INITIALIZATION ROUTINE
176

FIG. 17

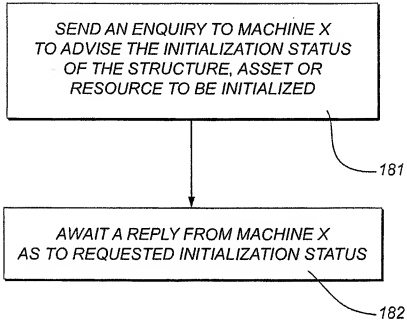
SEND AN ENQUIRY TO MACHINE X TO ADVISE THE INITIALIZATION STATUS OF THE STRUCTURE, ASSET OR RESOURCE TO BE INITIALIZED
181
AWAIT A REPLY FROM MACHINE X AS TO REQUESTED INITIALIZATION STATUS
182

FIG. 18

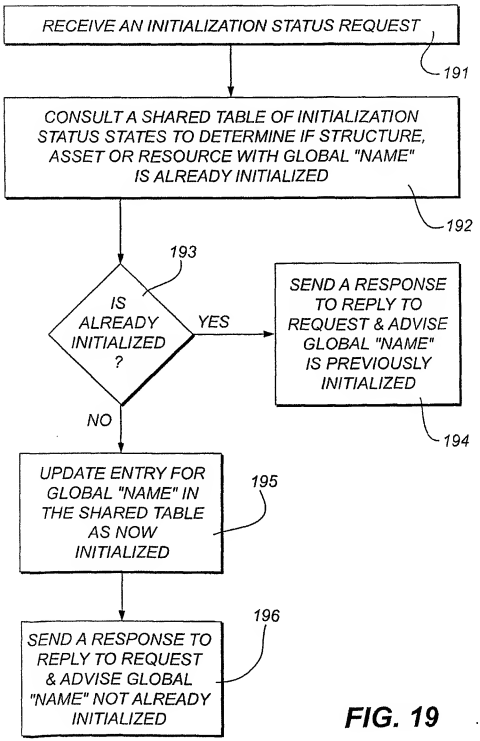
RECEIVE AN INITIALIZATION STATUS REQUEST
191
CONSULT A SHARED TABLE OF INITIALIZATION STATUS STATES TO DETERMINE IF STRUCTURE, ASSET OR RESOURCE WITH GLOBAL "NAME" IS ALREADY INITIALIZED
192
193
IS ALREADY INITIALIZED ?
YES
SEND A RESPONSE TO REPLY TO REQUEST & ADVISE GLOBAL "NAME" IS PREVIOUSLY INITIALIZED
194
NO
UPDATE ENTRY FOR GLOBAL "NAME" IN THE SHARED TABLE AS NOW INITIALIZED
195
SEND A RESPONSE TO REPLY TO REQUEST & ADVISE GLOBAL "NAME" NOT ALREADY INITIALIZED
196

FIG. 19

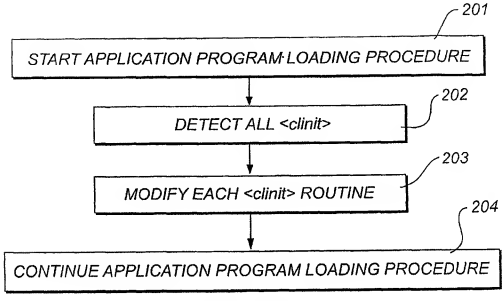
201
START APPLICATION PROGRAM LOADING PROCEDURE
202
DETECT ALL <clinit>
203
MODIFY EACH <clinit> ROUTINE
204
CONTINUE APPLICATION PROGRAM LOADING PROCEDURE

FIG. 20

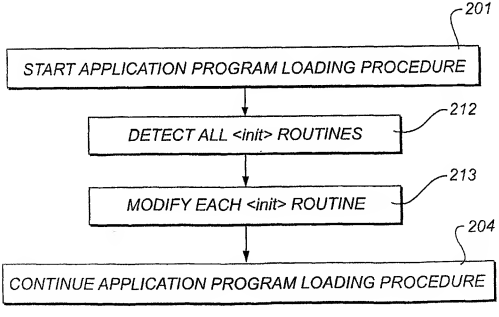
201
START APPLICATION PROGRAM LOADING PROCEDURE
212
DETECT ALL <init> ROUTINES
213
MODIFY EACH <init> ROUTINE
204
CONTINUE APPLICATION PROGRAM LOADING PROCEDURE

FIG. 21

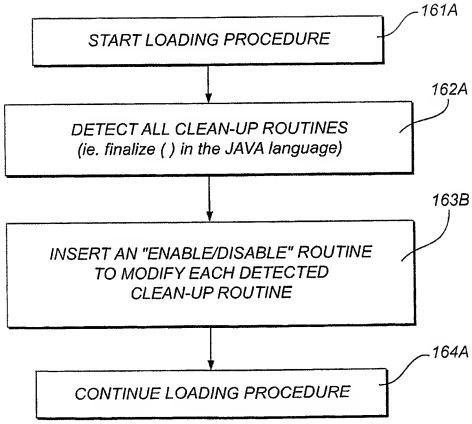
START LOADING PROCEDURE
161A
DETECT ALL CLEAN-UP ROUTINES
(ie. finalize ( ) in the JAVA language)
162A
INSERT AN "ENABLE/DISABLE" ROUTINE
TO MODIFY EACH DETECTED
CLEAN-UP ROUTINE
163B
CONTINUE LOADING PROCEDURE
164A

FIG. 22

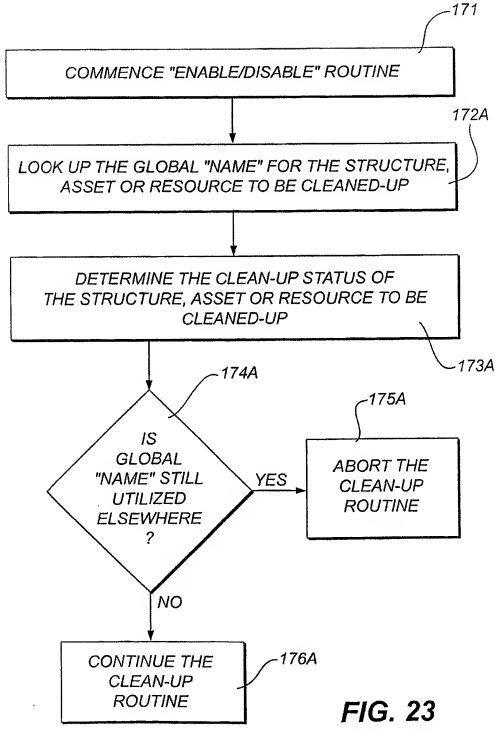
171
COMMENCE "ENABLE/DISABLE" ROUTINE
172A
LOOK UP THE GLOBAL "NAME" FOR THE STRUCTURE, ASSET OR RESOURCE TO BE CLEANED-UP
DETERMINE THE CLEAN-UP STATUS OF THE STRUCTURE, ASSET OR RESOURCE TO BE CLEANED-UP
173A
174A
IS GLOBAL "NAME" STILL UTILIZED ELSEWHERE ?
YES
175A
ABORT THE CLEAN-UP ROUTINE
NO
CONTINUE THE CLEAN-UP ROUTINE
176A

FIG. 23

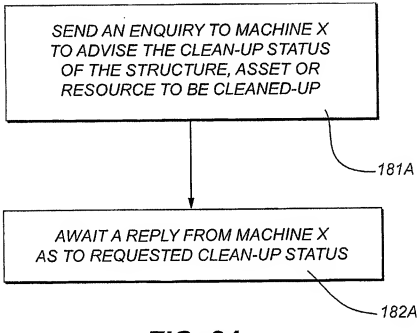
SEND AN ENQUIRY TO MACHINE X TO ADVISE THE CLEAN-UP STATUS OF THE STRUCTURE, ASSET OR RESOURCE TO BE CLEANED-UP
181A
AWAIT A REPLY FROM MACHINE X AS TO REQUESTED CLEAN-UP STATUS
182A

FIG. 24

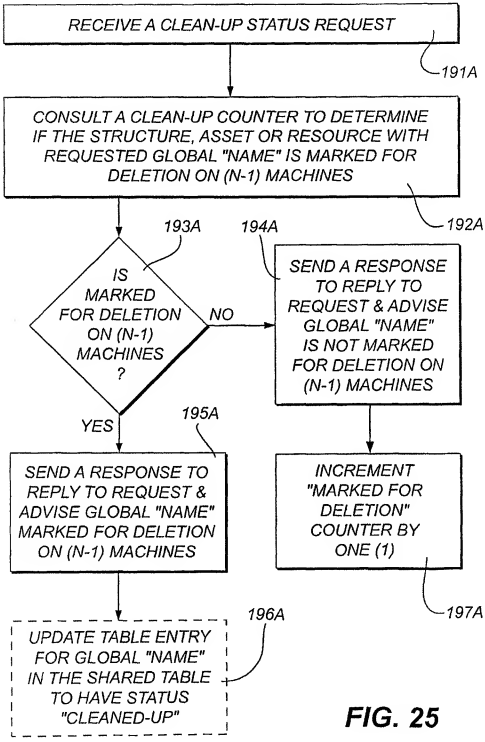
RECEIVE A CLEAN-UP STATUS REQUEST
191A
CONSULT A CLEAN-UP COUNTER TO DETERMINE IF THE STRUCTURE, ASSET OR RESOURCE WITH REQUESTED GLOBAL "NAME" IS MARKED FOR DELETION ON (N-1) MACHINES
192A
193A
IS MARKED FOR DELETION ON (N-1) MACHINES ?
NO
194A
SEND A RESPONSE TO REPLY TO REQUEST & ADVISE GLOBAL "NAME" IS NOT MARKED FOR DELETION ON (N-1) MACHINES
YES
195A
SEND A RESPONSE TO REPLY TO REQUEST & ADVISE GLOBAL "NAME" MARKED FOR DELETION ON (N-1) MACHINES
INCREMENT "MARKED FOR DELETION" COUNTER BY ONE (1)
197A
196A
UPDATE TABLE ENTRY FOR GLOBAL "NAME" IN THE SHARED TABLE TO HAVE STATUS "CLEANED-UP"

FIG. 25

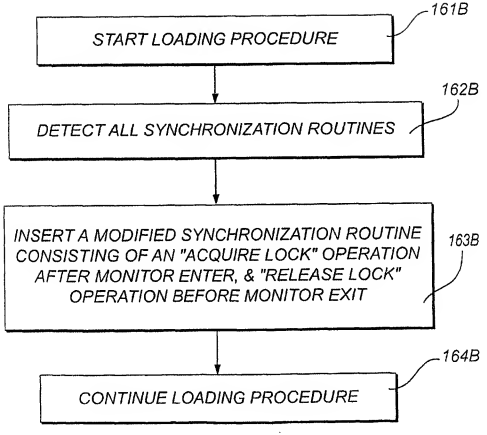
START LOADING PROCEDURE
161B
DETECT ALL SYNCHRONIZATION ROUTINES
162B
INSERT A MODIFIED SYNCHRONIZATION ROUTINE CONSISTING OF AN "ACQUIRE LOCK" OPERATION AFTER MONITOR ENTER, & "RELEASE LOCK" OPERATION BEFORE MONITOR EXIT
163B
CONTINUE LOADING PROCEDURE
164B

FIG. 26

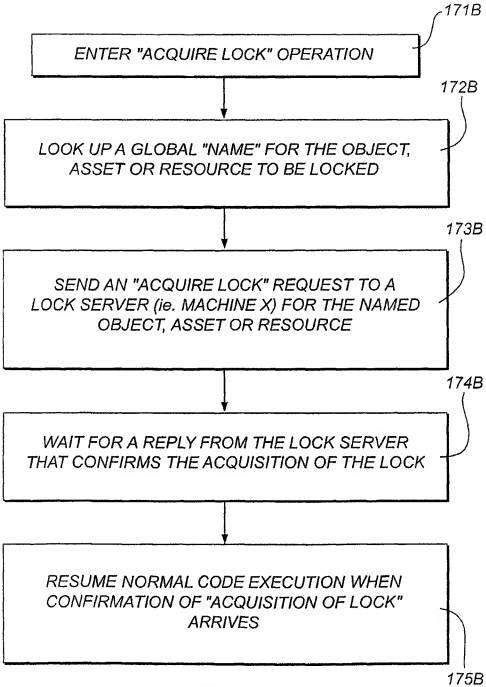
171B
ENTER "ACQUIRE LOCK" OPERATION
172B
LOOK UP A GLOBAL "NAME" FOR THE OBJECT, ASSET OR RESOURCE TO BE LOCKED
173B
SEND AN "ACQUIRE LOCK" REQUEST TO A LOCK SERVER (ie. MACHINE X) FOR THE NAMED OBJECT, ASSET OR RESOURCE
174B
WAIT FOR A REPLY FROM THE LOCK SERVER THAT CONFIRMS THE ACQUISITION OF THE LOCK
RESUME NORMAL CODE EXECUTION WHEN CONFIRMATION OF "ACQUISITION OF LOCK" ARRIVES
175B

FIG. 27

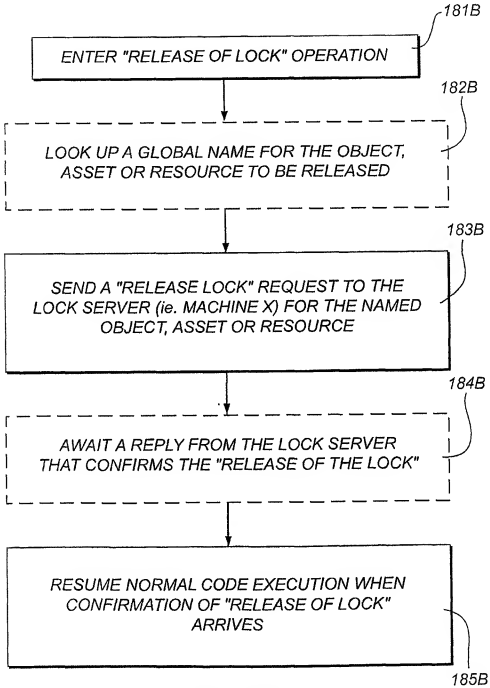
181B
ENTER "RELEASE OF LOCK" OPERATION
182B
LOOK UP A GLOBAL NAME FOR THE OBJECT, ASSET OR RESOURCE TO BE RELEASED
183B
SEND A "RELEASE LOCK" REQUEST TO THE LOCK SERVER (ie. MACHINE X) FOR THE NAMED OBJECT, ASSET OR RESOURCE
184B
AWAIT A REPLY FROM THE LOCK SERVER THAT CONFIRMS THE "RELEASE OF THE LOCK"
RESUME NORMAL CODE EXECUTION WHEN CONFIRMATION OF "RELEASE OF LOCK" ARRIVES
185B

FIG. 28

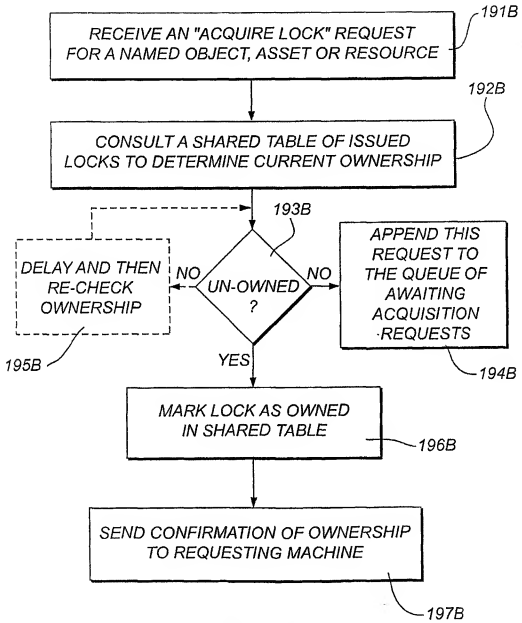
RECEIVE AN "ACQUIRE LOCK" REQUEST FOR A NAMED OBJECT, ASSET OR RESOURCE
191B
CONSULT A SHARED TABLE OF ISSUED LOCKS TO DETERMINE CURRENT OWNERSHIP
192B
193B
UN-OWNED ?
NO
NO
YES
DELAY AND THEN RE-CHECK OWNERSHIP
195B
APPEND THIS REQUEST TO THE QUEUE OF AWAITING ACQUISITION REQUESTS
194B
MARK LOCK AS OWNED IN SHARED TABLE
196B
SEND CONFIRMATION OF OWNERSHIP TO REQUESTING MACHINE
197B

FIG. 29

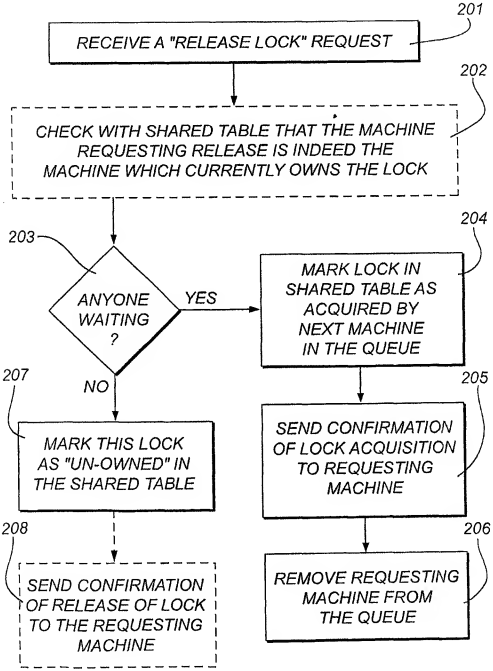
201
RECEIVE A "RELEASE LOCK" REQUEST
202
CHECK WITH SHARED TABLE THAT THE MACHINE REQUESTING RELEASE IS INDEED THE MACHINE WHICH CURRENTLY OWNS THE LOCK
203
ANYONE WAITING ?
YES
NO
204
MARK LOCK IN SHARED TABLE AS ACQUIRED BY NEXT MACHINE IN THE QUEUE
205
SEND CONFIRMATION OF LOCK ACQUISITION TO REQUESTING MACHINE
206
REMOVE REQUESTING MACHINE FROM THE QUEUE
207
MARK THIS LOCK AS "UN-OWNED" IN THE SHARED TABLE
208
SEND CONFIRMATION OF RELEASE OF LOCK TO THE REQUESTING MACHINE

FIG. 30

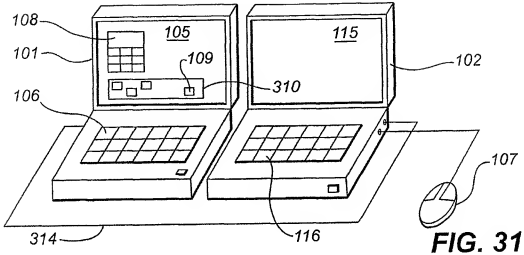

FIG. 31

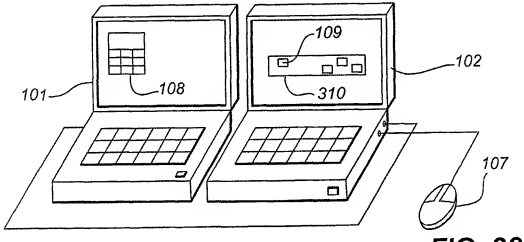

FIG. 32

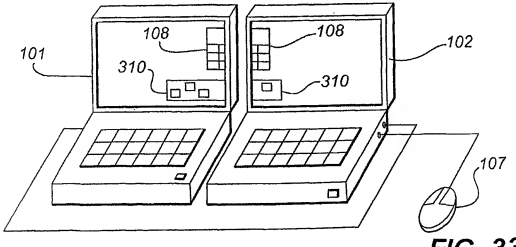

FIG. 33

INTERNATIONAL SEARCH REPORT

International application No.
PCT/AU2005/001641

A. CLASSIFICATION OF SUBJECT MATTER

Int. Cl.

G06F 15/16 (2006.01)

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
DWPI, USPTO, PCT, IEEE, internet (object, replicate, redundant, mirror, cluster, distributed, lock, thread, compile, runtime, virtual machine, etc. etc.)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	WO 2003/083614 A1 (ETERNAL SYSTEMS, INC.), 9 October 2003 the whole document	1-163
A	US 6,625,751 B1 (STAROVIC et al), 23 September 2003 the whole document	1-163
A	WO 2002/044835 A2 (GINGERICH), 6 June 2002 the whole document	1-163
A	US 2004/0073828 A1 (BRONSTEIN), 15 April 2004 the whole document	1-163

[X] Further documents are listed in the continuation of Box C [X] See patent family annex

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
9 December 2005	22 DEC 2005
Name and mailing address of the ISA/AU	Authorized officer
AUSTRALIAN PATENT OFFICE PO BOX 200, WODEN ACT 2606, AUSTRALIA E-mail address: pct@ipaustralia.gov.au Facsimile No. (02) 6285 3929	**Matthew Hollingworth** Telephone No : (02) 6283 2024

Form PCT/ISA/210 (second sheet) (April 2005)

INTERNATIONAL SEARCH REPORT

International application No.
PCT/AU2005/001641

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5,488,723 A (BARADEL et al), 30 January 1996 the whole document	1-163
A	T. C. Bressoud, *TFT: A Software System for Application-Transparent Fault Tolerance* Proc. 28th Annual International Symposium on Fault-Tolerant Computing, pp. 128-37, 1998	1-163
A	H. E. Bal et al, *Orca: A Language for Parallel Programming of Distributed Systems* IEEE Transactions on Software Engineering, 18(3), pp. 190-205, March 1992	1-163
X, E	US 2005/0240737 A1 (HOLT), 27 October 2005 the whole document	1-163
X, E	WO 2005/103928 A1 (WARATEK PTY LTD), 3 November 2005 the whole document	1-163

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.
PCT/AU2005/001641

This Annex lists the known "A" publication level patent family members relating to the patent documents cited in the above-mentioned international search report. The Australian Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

Patent Document Cited in Search Report		Patent Family Member					
WO	03083614	AU	2003223352	AU	2003230748	EP	1495414
		EP	1495571	US	2004078617	US	2004078618
		WO	03084116				
US	6625751	GB	2353113				
WO	0244835	AU	32411/02				
US	2004073828	NONE					
US	5488723	EP	0572307	FR	2691559	WO	9324884
US	2005240737	US	2005257219	US	2005262313	US	2005262513
WO	2005103928	WO	2005103924	WO	2005103925	WO	2005103926
		WO	2005103927				

Due to data integration issues this family listing may not include 10 digit Australian applications filed since May 2001.

END OF ANNEX